4. Employee Management System

Understanding Array Representation

Arrays in Memory:

Contiguous Memory Allocation: Arrays are stored in contiguous memory locations. This means that if the array starts at memory address x, and each element occupies k bytes, the elements of the array are located at addresses x, x + k, x + 2k, and so on.

Advantages:

- **Fast Access**: Arrays provide O(1) time complexity for accessing elements by index. This means you can retrieve any element directly if you know its index.
- **Memory Efficiency**: Arrays have a fixed size, which allows for efficient memory allocation and deallocation since the memory is allocated in a single block.
- **Cache-Friendly**: Due to contiguous memory allocation, arrays are more cache-friendly, leading to better performance in terms of access speed. Sequential access of array elements benefits from spatial locality.

Analysis: Time Complexity and Limitations of Arrays

Time Complexity:

- **Add (at the end)**: O(1) if there's space. O(n) if resizing is needed (dynamic arrays).
- **Search**: O(n) for unsorted arrays (linear search), O(log n) for sorted arrays (binary search).
- **Traverse**: O(n), as each element is accessed once.
- **Delete**: O(n) in the worst case, as elements may need to be shifted to fill the gap.

Limitations of Arrays:

- Once an array is allocated, its size cannot be changed**(fixed size).** If you need a dynamically sized collection, you might need to use a dynamic array (e.g., ArrayList in Java) or another data structure like a linked list.
- Inserting or deleting elements (other than at the end) requires shifting elements, leading to **O(n)** time complexity. This can be **inefficient** for large datasets.
- If the array is not fully utilized, it leads to **wasted memory**. Conversely, if the array needs to grow, it requires copying elements to a new, larger array, which is time-consuming.

- Arrays are **not suitable** for scenarios where frequent **random insertions and deletions** are required **(Sequential Access)** . Linked lists or other dynamic data structures might be more appropriate in such cases.

When to Use Arrays:

- When the size of the dataset is known and fixed.
- When fast access to elements by index is required.
- When memory overhead needs to be minimized.
- When the operations are mostly traversals or accessing elements by index, and not frequent insertions or deletions.